CMS Event Processing Multi-core Efficiency Status

Dr Christopher Jones *on behalf of CMS Offline and Computing*

CHEP 2016

10 October 2016

Multi-threading Use In CMS

Multiple threads used for standard work

Have run reconstruction step with 4 threads

- prompt data usage for all LHC Run 2
- Monte Carlo production since Summer 2016

Running HLT 4 with threads since September of 2015

Simulation step with 4 threads will start Winter 2016

- Capability has been tested since Summer 2015

Only supported having one thread per concurrent event

Referred to in this talk as **Original**

Only using 4 threads per job since

Has good CPU efficiency

Sufficient to staying within memory limits of worker nodes

Keeps the number of simultaneous jobs controlled by workflow management to a workable limit

Processing Stalls

Sharing resources across concurrently running events leads to thread stalls

Examples

All reads from ROOT input file must be serialized

Writing to a ROOT output file must be serialized

- Note: can write to different ROOT output files simultaneously

Legacy modules are not thread safe so the framework will only run one at a time

One thread per event implementation ran modules in fixed order

Could not schedule around algorithm waiting for a shared resource

Stall Demonstration

Simple configuration to demonstrate stalls

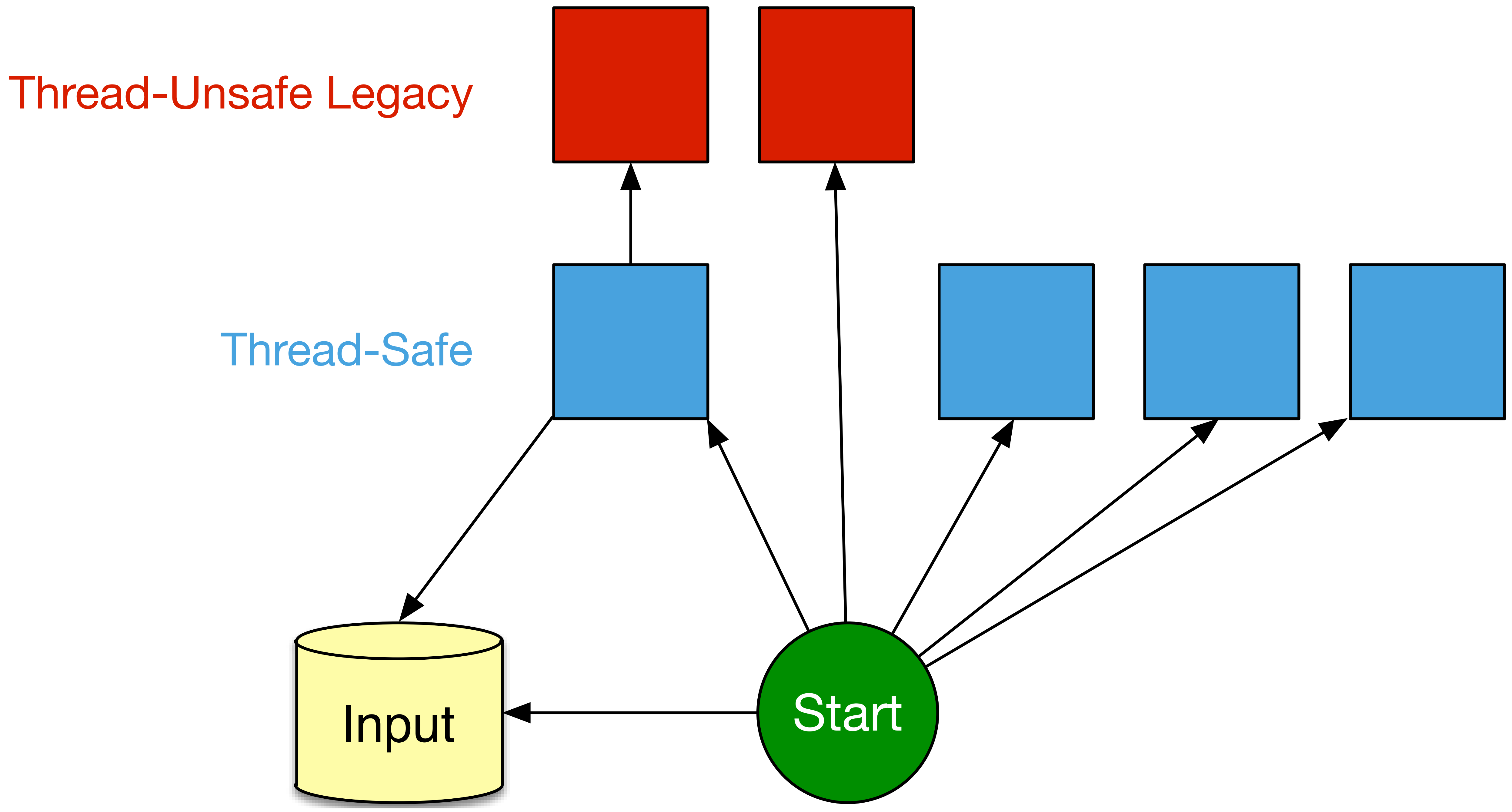
Reads from input file

Two legacy modules which cannot run simultaneously

Five additional modules which are thread-safe

Data dependencies between modules constrain allowed concurrency

- Note: modules wait until all their data has been made available from other modules



Example Stall with Original Implementation

Four concurrent event loops

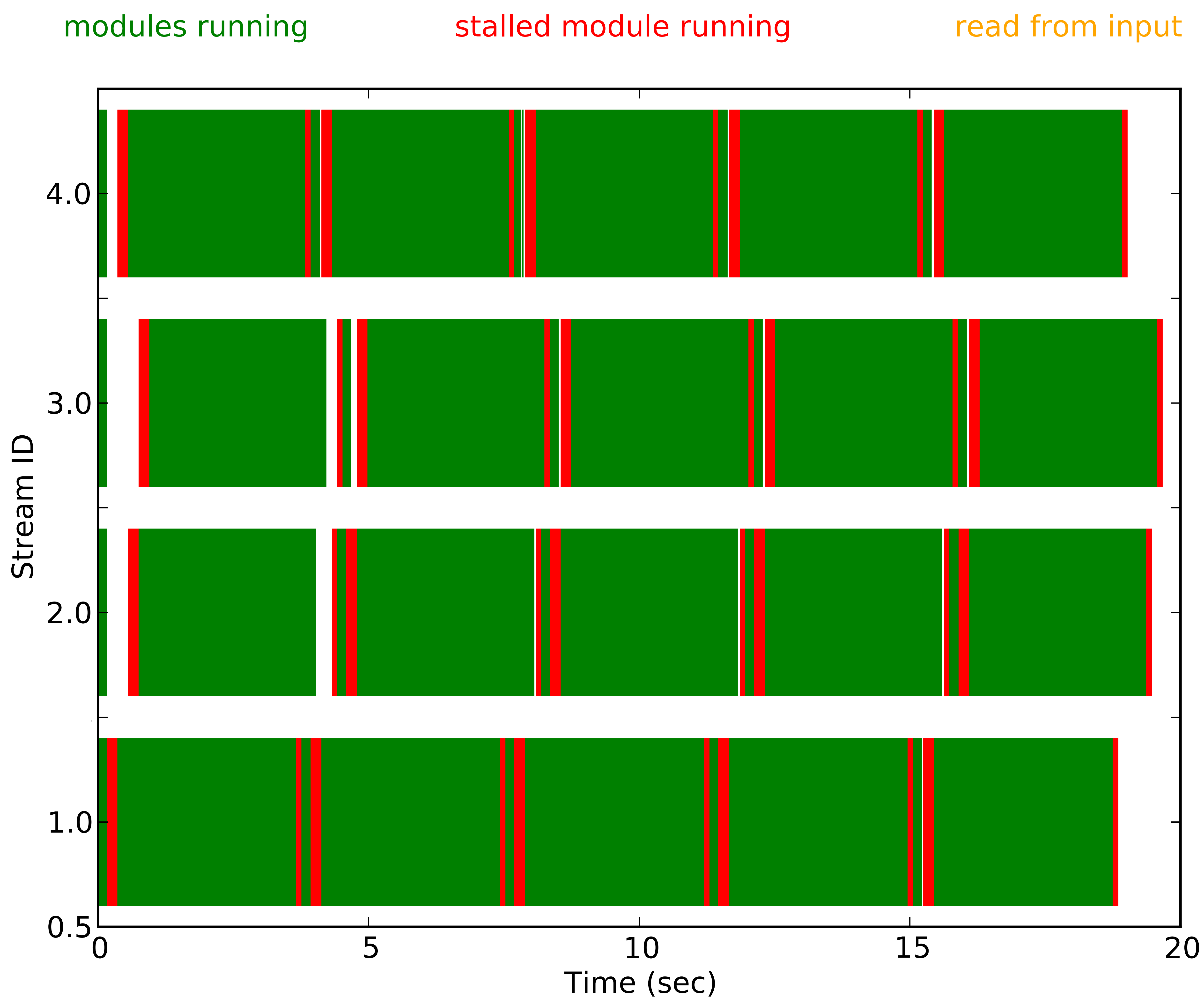
- Each loop is referred to as a **stream**
- Each loop can only use one thread

Green is when a module is running in stream

White is when no module running in stream

Red is when a stalled module is running

- **White precedes red when a stall happens**
- **Module stalls because it can not run concurrently and another stream is running the module**



Stall Mitigation using Multiple Threads per Concurrent Event (1)

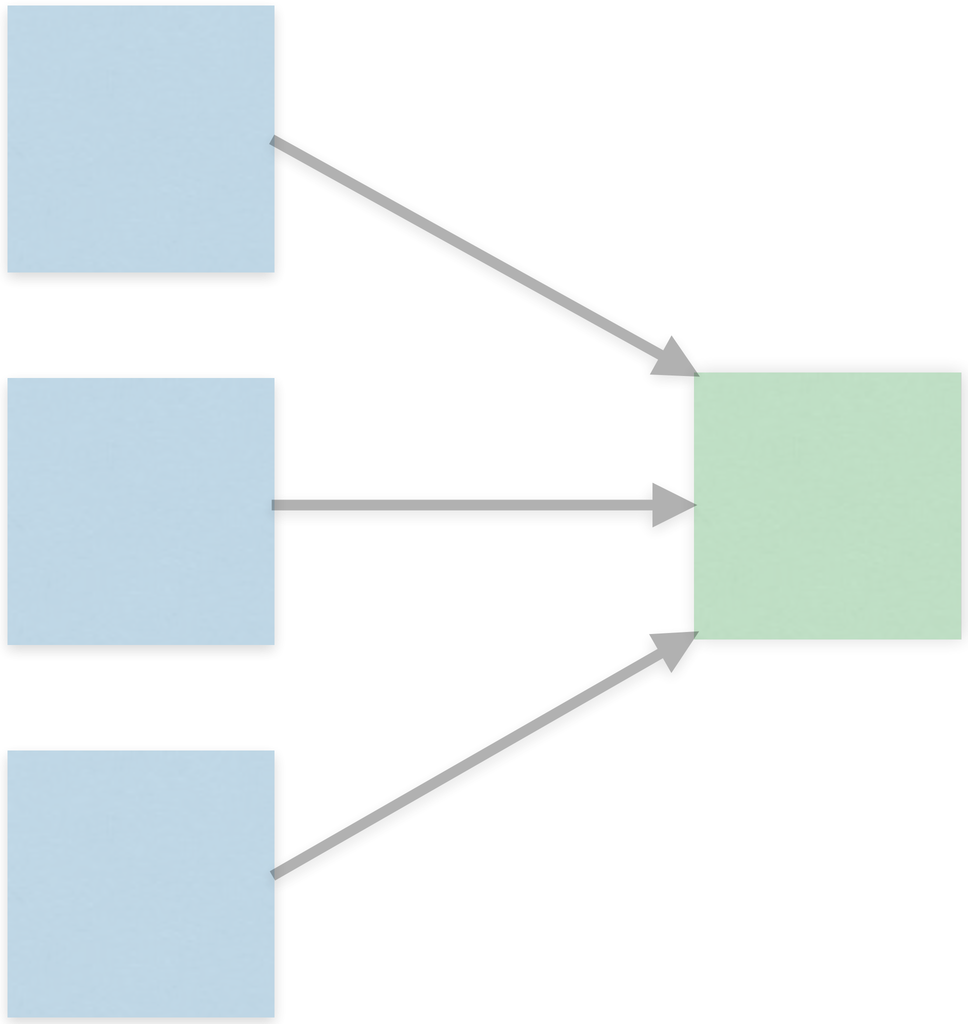
All independent sequences of event filtering modules are started simultaneously

Within a sequence the modules must be run within the set order

Stall Mitigation using Multiple Threads per Concurrent Event (2)

Data for modules are prefetched asynchronously

- Provides a large number of tasks for TBB to schedule
- Module starts after prefetches have finished

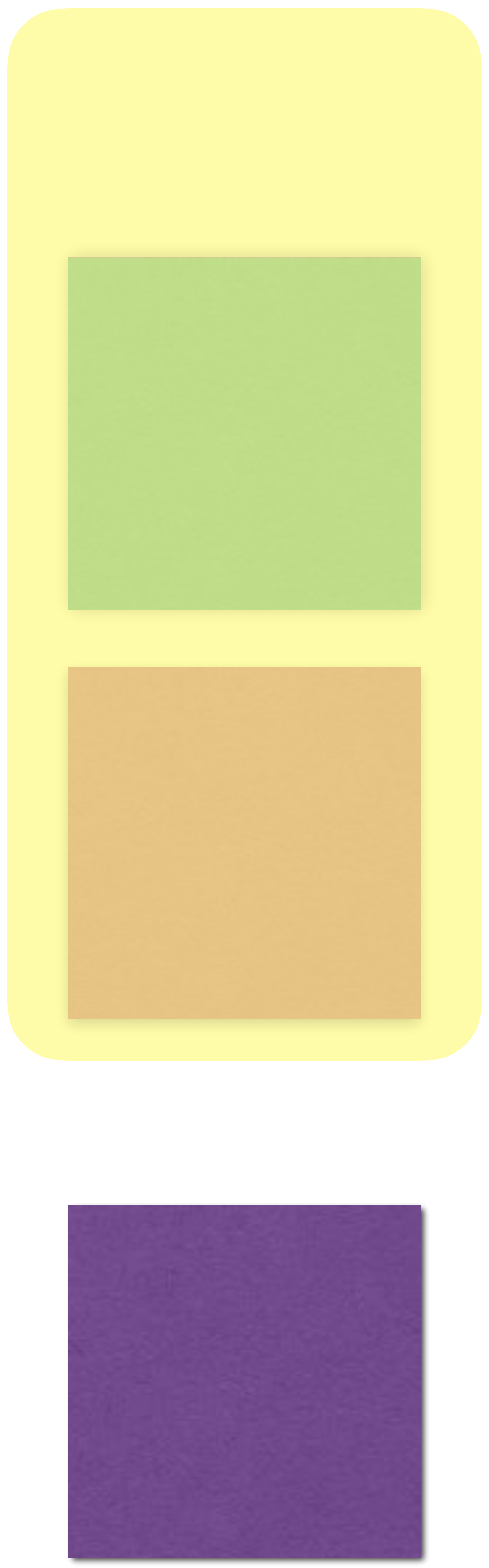


Stall Mitigation using Multiple Threads per Concurrent Event (3)

A shared resource is guarded by a serial task queue

Modules needing the resource have their ‘to run’ task placed in the appropriate queue

When a task from a queue finishes, it automatically starts the next task in the queue

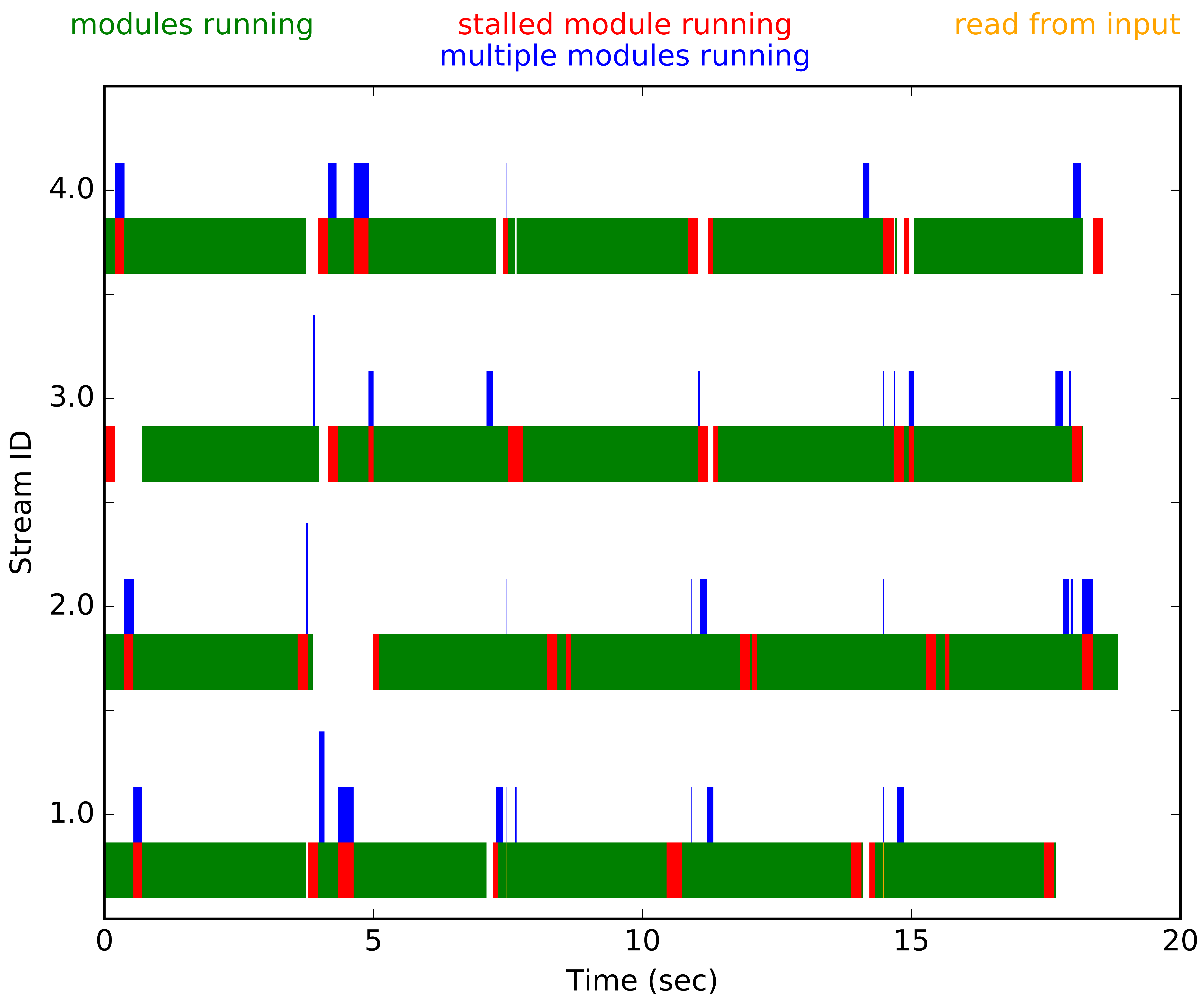


Four threads with four streams

Blue when multiple modules running in a stream

Height of blue bar proportional to number of running modules

Blue on one stream corresponds to white on another

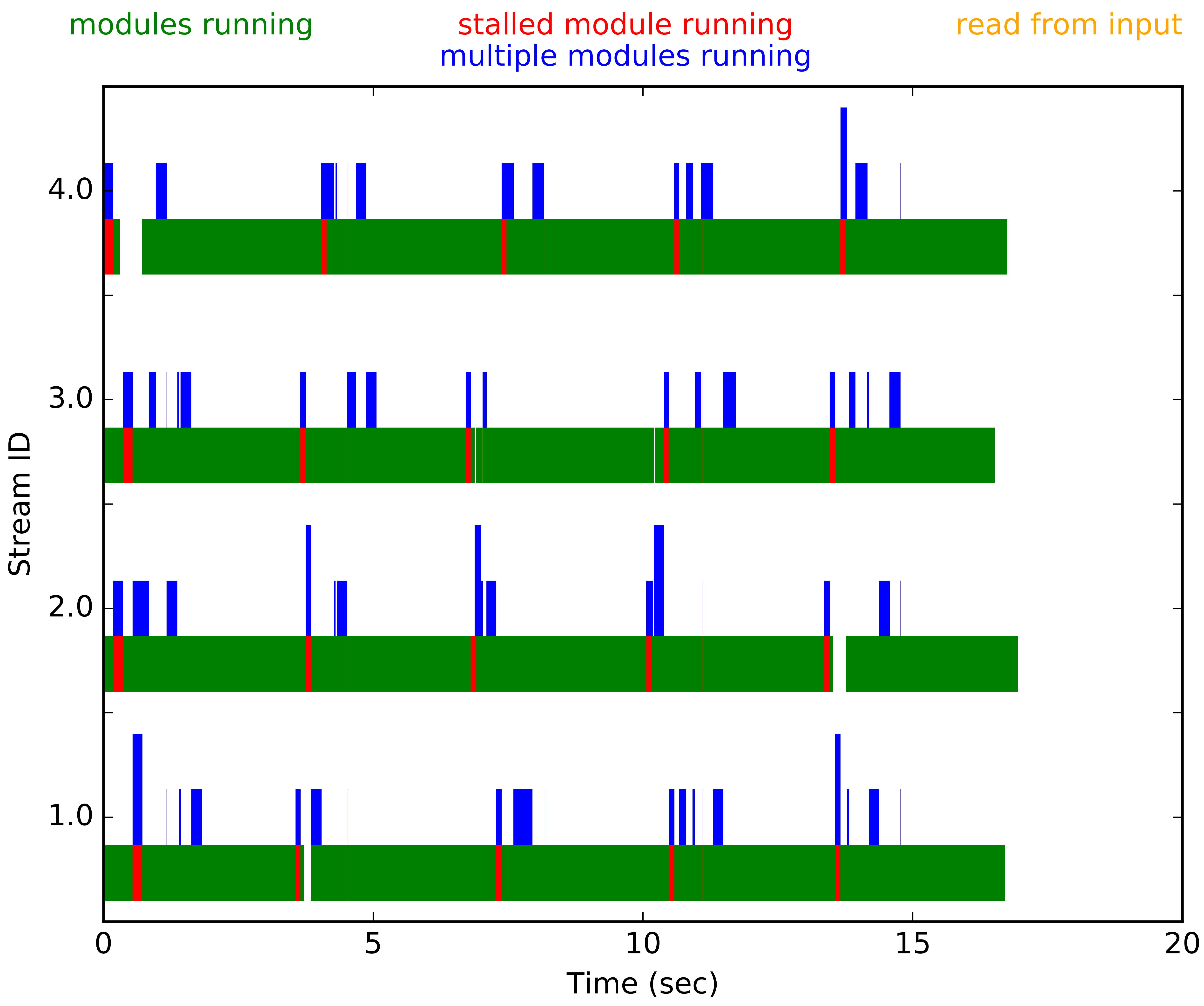


Example Stall Mitigation with Multiple Threads

Five threads with four streams

Stalls greatly mitigated

- Job finishes in less time



Fermilab

Realistic Demonstration Measurements

Machine for testing

Westmere-EP L5640 CPU with 6 cores x 2 hyper-threads

Compared Reconstruction jobs

Original one-thread-per-event

Concurrent modules per event with number threads == number of streams

Concurrent modules per event with number threads == 12

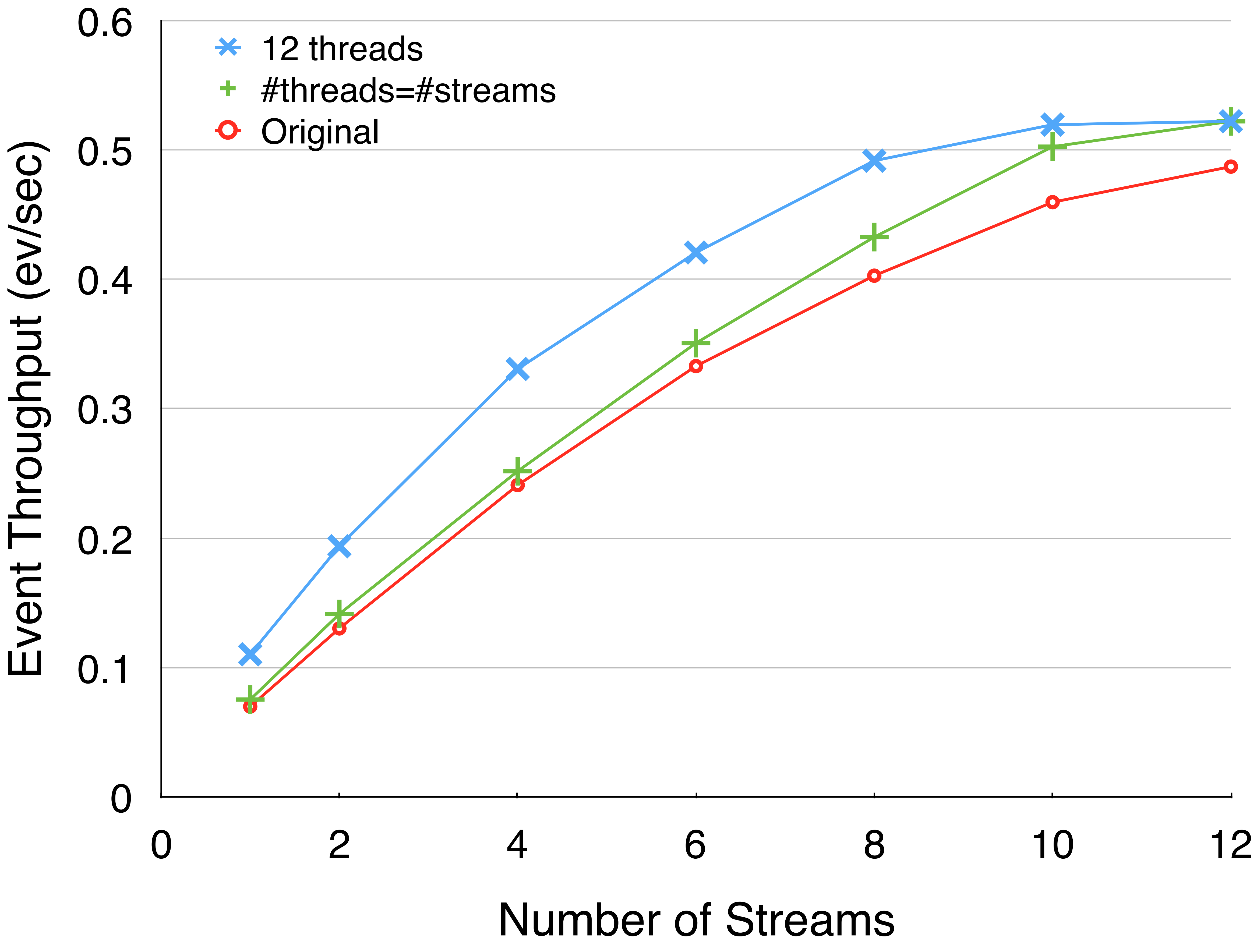
Reconstruction configuration summary

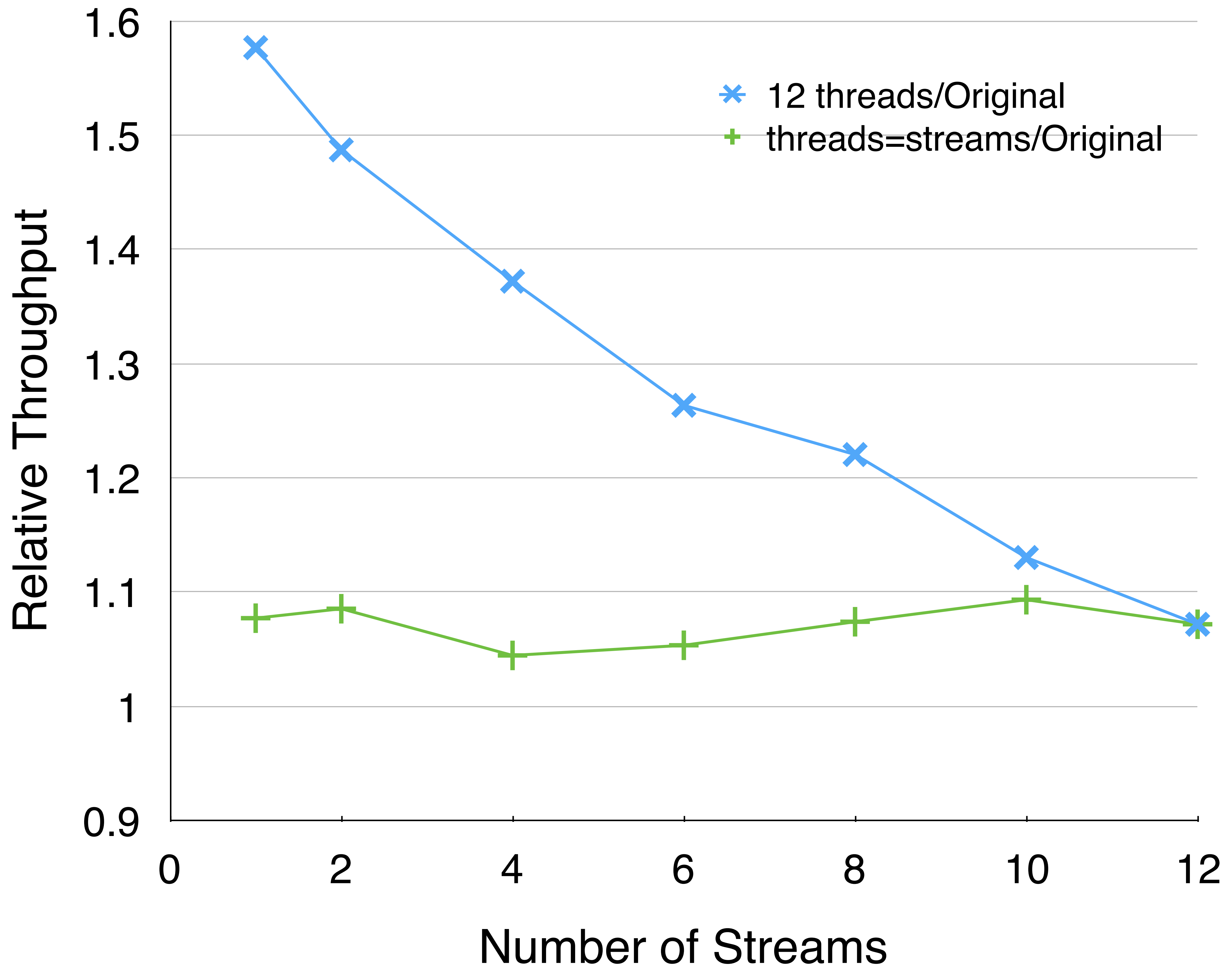
3 output modules

1780 other modules

21 filter sequences

Event Throughput Comparison





Reconstruction with 8 Threads and 6 Concurrent Events

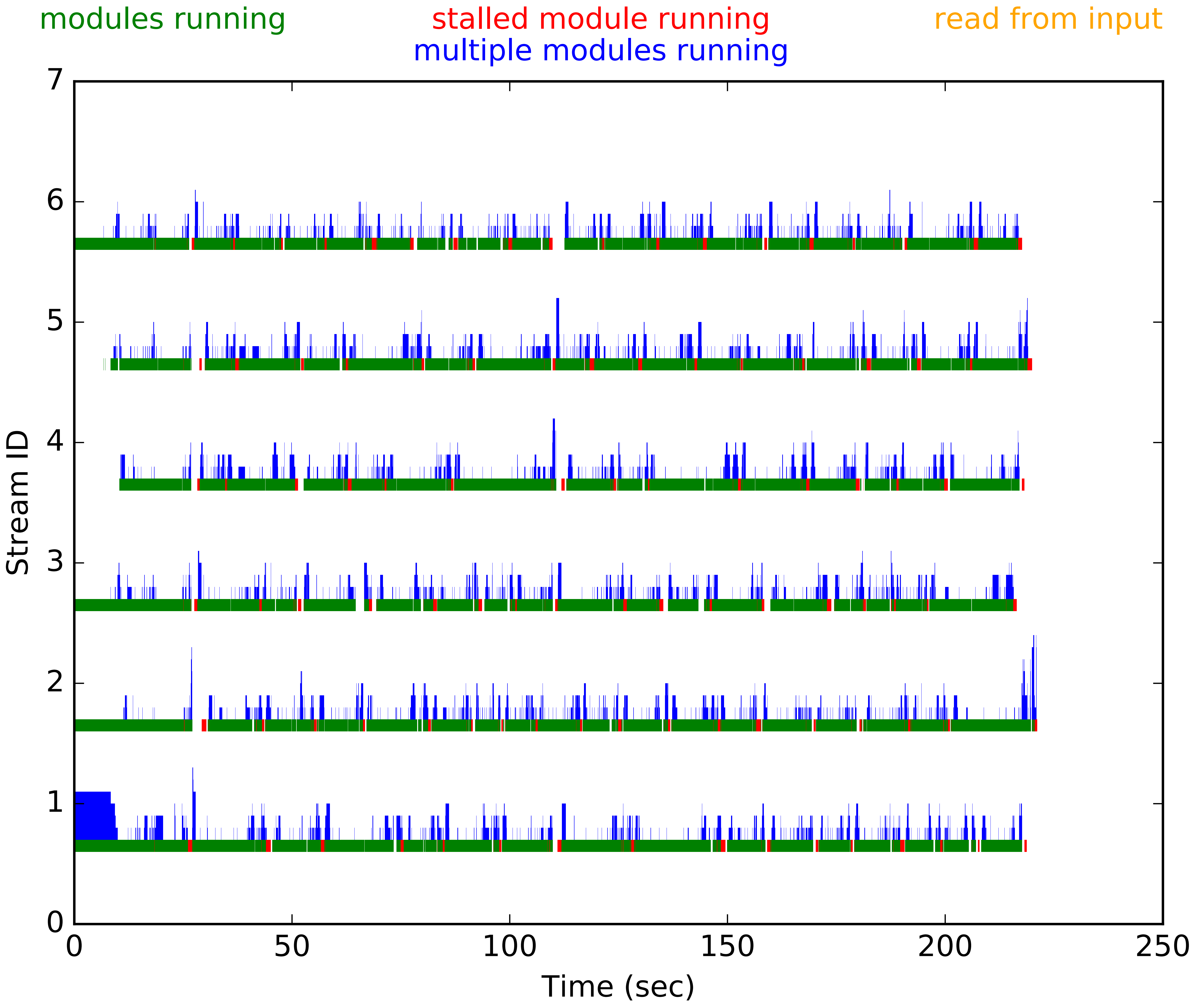
Stalls are solely caused by one output module

- The one which takes longest per event

Dynamic scheduling allows stall mitigation

- Can reorder legacy modules and other output modules

Additional threads increase throughput



Memory Utilization per Stream

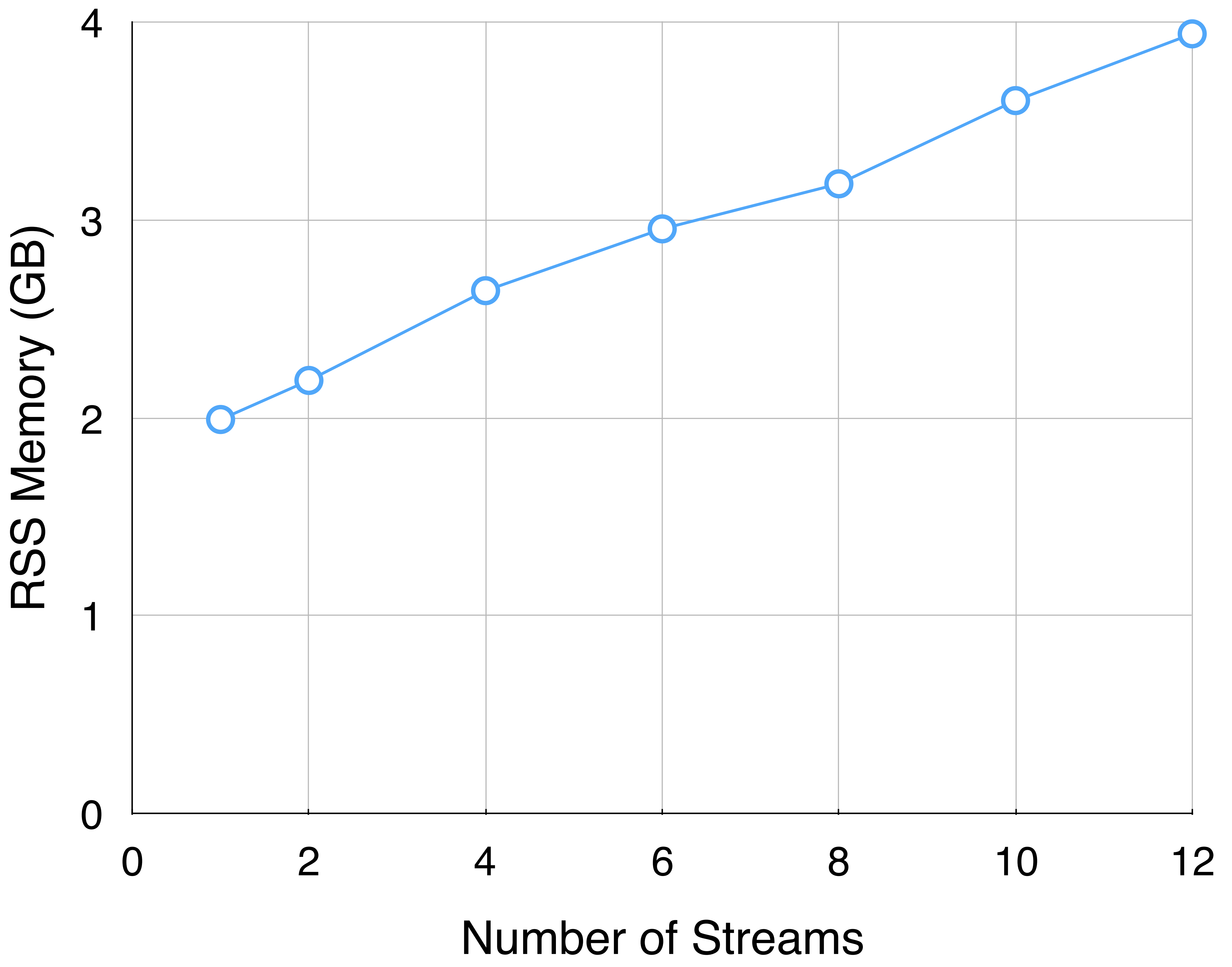
High initial memory

- ~ 2 GB

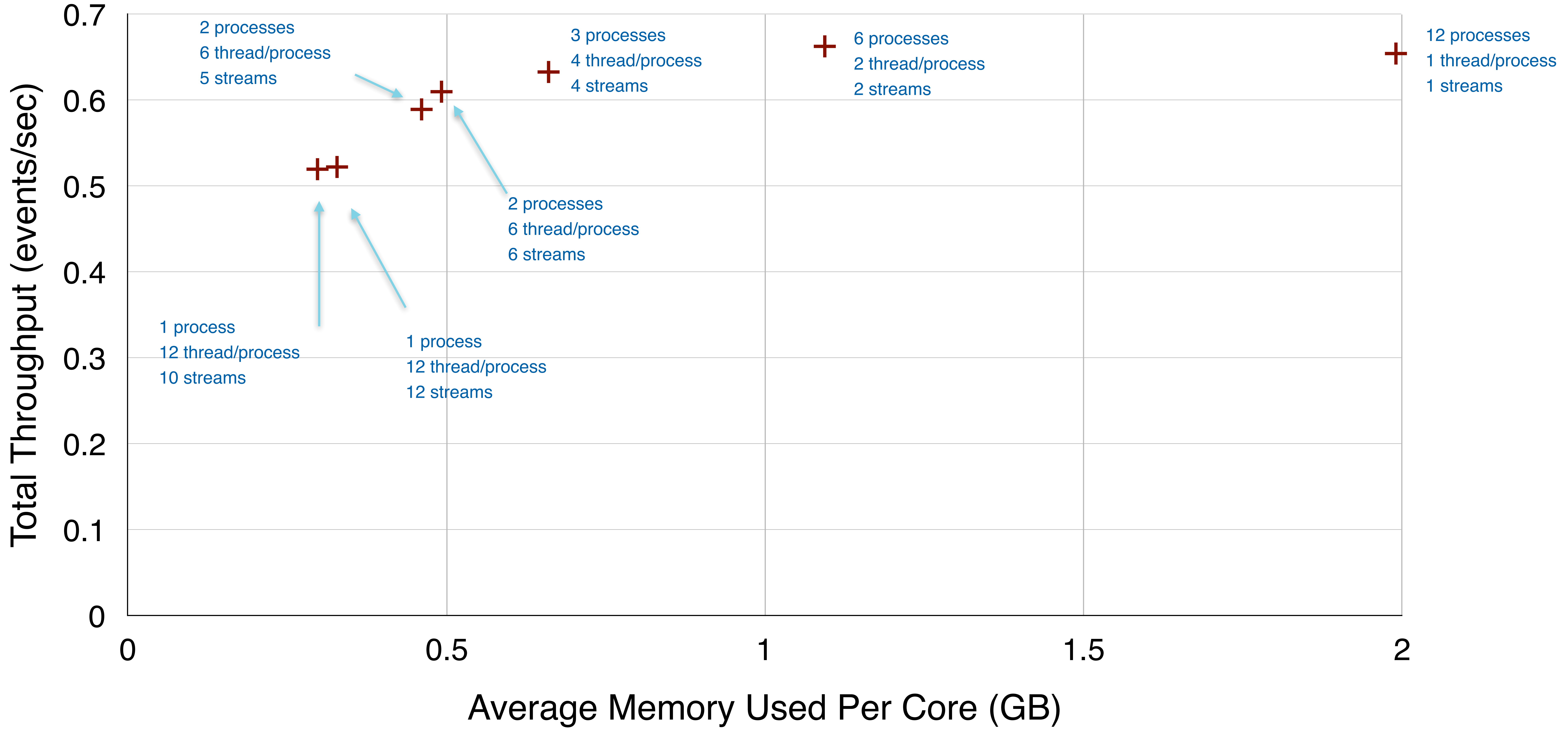
Memory grows slowly w.r.t number of streams

- ~150 MB/stream

Increasing number of threads does not noticeably increase memory usage



Total Throughput vs Memory for Fully Loaded Machine



Choose (number of processes) * (number of threads) to utilize all twelve cores

Included number of threads > number of streams

Can choose reasonable options between 2 ,1 and .5 GB/core options

Fermilab

Conclusion

CMS has successfully utilized multi-threaded processing jobs

- All prompt reconstruction for Run 2 were multi-threaded jobs

Allowing multiple threads per event will allow

- processing of more memory intensive jobs
- utilization of machines with lower memory per core limits

Greater threading efficiency is important as CMS is continually increases its utilization of multi-threading